Fast parallel remeshing for accurate large-eddy simulations on very large meshes

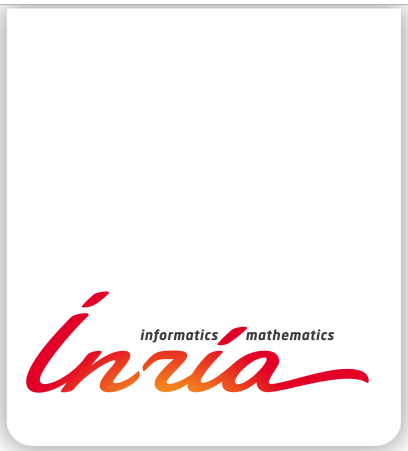
Cédric Lachat, François Pellegrini, Cécile Dobrzynski, Gabriel Staffelbach

Fast parallel remeshing for accurate large-eddy simulations on very large meshes

Cédric Lachat, François Pellegrini, Cécile Dobrzynski, Gabriel Staffelbach

RESEARCH REPORT
N° 9133
December 2017
Project-Teams Tadaam

ISSN 0249-6399 ISRN INRIA/RR--9133--FR+ENG



Fast parallel remeshing for accurate large-eddy simulations on very large meshes

Cédric Lachat*, François Pellegrini†, Cécile Dobrzynski‡,

Gabriel Staffelbach§

Project-Teams Tadaam

Research Report n° 9133 — December 2017 — 13 pages

Abstract: Numerical simulations on very large meshes, such as large-addy simulations (LES), cannot be performed without resorting to distributed-memory parallelism. For these methods, a sufficient precision can only be achieved by remeshing dynamically the areas that need it. Such a remeshing must therefore be performed in parallel. This paper presents the coarse-grain parallel remeshing method which has been devised and implemented in the PaMPA library for handling distributed meshes in parallel. This method is validated in the context of an industrial LES simulation on a helicopter turbine combustion chamber, on a mesh of more than one billion elements.

Key-words: distributed mesh, parallel remeshing, large-eddy simulation, LES

* Inria Bordeaux – Sud-Ouest, funded by ADT "*El Gaucho*" and PIA ELCI.
† Université de Bordeaux, LaBRI & Inria Bordeaux – Sud-Ouest.
‡ Institut Polytechnique de Bordeaux, IMB & Inria – Bordeaux Sud-Ouest.
§ CERFACS : Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique.

RESEARCH CENTRE
BORDEAUX – SUD-OUEST

200 avenue de la Vieille Tour
33405 Talence Cedex

Remaillage parallèle rapide pour les simulations de grands écoulements (LES) sur des maillages de très grande taille

Résumé : Les simulations numériques portant sur des maillages de très grande taille, telles que les méthodes LES (« *large-eddy simulations* »), ne peuvent être réalisées qu'en ayant recours au parallélisme à mémoire distribuée. Pour ces méthodes, une précision suffisante ne peut être atteinte qu'en remaillant dynamiquement les zones qui le nécessitent. Ce remaillage doit donc être effectué en parallèle. Cet article présente la méthode de remaillage parallèle à gros grain conçue et mise en œuvre au sein de la bibliothèque PaMPA de gestion parallèle de maillages distribués. Cette méthode est validée dans le cadre d'une simulation LES industrielle de chambre de combustion de turbine d'hélicoptère, portant sur un maillage à plus d'un milliard d'éléments.

Mots-clés : maillage distribué, remaillage parallèle, méthode LES

Contents

1 Introduction

Parallel remeshing methods can be categorized into two major families. The first one, the study of which started by the end of the 1990s, is that of fine-grained parallel methods. They aim at performing concurrently the elementary remeshing primitives, such as element subdivision, edge flipping, or node insertion or removal, on the same mesh. However, such a fine-grained parallelism is difficult to implement, due to the amount of fine-grained synchronization and locking that must be implemented in order to preserve mesh consistency when two neighboring elements are processed concurrently. Moreover, it implies using shared-memory parallelism, which is not scalable for a large number of processing elements. The second family, the study of which started in the early 2000s, overcomes these issues by considering coarse-grain parallelism only. Larger, non-overlapping areas of the mesh are defined, on which existing sequential remeshing libraries can be applied concurrently and independently. Such methods can be used either in a shared-memory, or a distributed-memory context.

We consider this second approach to be more scalable, that is, to allow for the processing of larger meshes on a larger number of processing elements, while preserving parallel efficiency. In order to validate this assumption, a comprehensive software framework for handling distributed meshes and using sequential remeshers in a distributed memory context had to be created. This was the purpose of Cédric Lachat's PhD thesis [13], which resulted in the creation of the PaMPA software [27]. This software has been designed to fulfill the needs of numerical solver writers, by providing relevant abstractions of mesh components. The algorithms it implements have been designed to scale up to handle mesh sizes above a billion elements, to be used in frontier simulations. This paper is structured as follows. Section 2 provides an overview of existing methods used to perform remeshing using several processing elements. Section 3 presents the method that has been devised and implemented in PaMPA. Section 4 evidences the efficiency of PaMPA in an industrial context, for performing a scientific simulation on meshes remeshed up to above one billion elements. Then comes the conclusion.

2 State of the art

Two different reasons may motivate the use of remeshing during a numerical simulation. The first one is to reduce numerical inaccuracy induced by an inadequate mesh. Most often, this occurs when the mesh is too coarse. Yet, a too fine mesh may induce numerical error as well, and wastes too much compute time. The second reason is a changing geometry: moving bodies, deformations, etc. Sequential remeshers face physical limits which hinder their practical use. The hardest limit is that of available memory per processing element (PE), but run time also becomes a strong concern for very large meshes. This is why many authors investigated parallel remeshing. Shared-memory parallelism is a straightforward way to reduce run time, at the expense of only small modifications to existing sequential remeshing algorithms, namely by adding locking mechanisms that prevent multiple PEs to process simultaneously the same or neighboring elements. However, shared-memory parallelism cannot scale to a large number of processing elements, because of memory bottlenecks. Moreover, all large-scale parallel architectures implement a distributed-memory model. Hence, on-the-fly remeshing of distributed meshes used in large-scale simulations requires distributed-memory parallel remeshing methods. In the case where the physical nodes of the parallel cluster comprise several cores, shared-memory parallelism can be used at the node level, but explicit message passing still has to be performed across machine nodes. There exist two main families of remeshing algorithms. The first one [4, 14, 22, 21, 7, 3, 16], which may seem the most natural, involves fine-grain parallelization of existing sequential remeshing primitives and methods, such as Delaunay triangulation [12], frontal methods [23] and edge subdivision [20]. However, in a distributed-memory context, synchronization across PEs amounts to exchanging messages across PEs which own neighboring elements, to make sure that, e.g., a node is not created on the same edge by several PEs at the same time. Such fine-grained communications usually lead to significant performance drops, due to the latency of the communication subsystem [18]. This is why a second family [8, 1, 11, 25, 5, 24, 17, 19, 2, 10, 28] of coarser-grain methods has been investigated, which uses existing sequential remeshers as black boxes within an iterative framework. Coarse-grained methods have strong benefits, but possess some drawbacks as well. While shared-memory, fine-grained methods implicitly balance the workload across PEs, which share the queue of elements to be remeshed, coarse-grained, distributed-memory methods require much more elaborate mechanisms to achieve parallel efficiency. For instance, the mesh has to be distributed across the PEs using some partitioning tool [18], mesh pieces of equivalent workload have to be assigned to the PEs [25] (which requires to estimate *a priori* the remeshing workload of a piece), and the number of pieces should match the number of PEs [6, 26], among other constraints. PaMPA pushes these ideas further, by implementing them in a generic and scalable framework for handling distributed meshes of any types of elements, as will be presented in the next section.

3 Parallel remeshing in PaMPA

As we have seen, due to technological limitations and especially memory bottlenecks, parallel remeshing on a large number of PEs can only be addressed using distributed-memory parallelism. Also, users may want to keep using the parallel remeshing library that they are used to. Hence, we have devised a coarse-grained parallel framework for taking advantage of existing sequential remeshers in a distributed-memory context.

3.1 Our method

Our method for parallel remeshing is an iterative algorithm made of two nested loops. The outer loops iterates the remeshing process until all elements satisfy the user-provided quality criterion. It starts by computing the quality of all elements according to the user-provided metric. All elements that do not respect this quality criterion are flagged for remeshing. The criterion may involve metrics such as smallest or largest edge length, or element geometry (quality), possibly considering anisotropy requirements. Then, the sub-mesh comprising all the tagged elements is passed to the inner loop. The outer loop stops when an inner loop does not bring enough improvement to the mesh, that is, when the ratio of tagged elements between two iterations of the outer loop becomes close to 1.

The inner loop, depicted in Figure 1, comprises five consecutive steps. It splits the groups of elements to be remeshed into pieces that will be remeshed concurrently on as many PEs as possible by a third-party, sequential remesher. Unlike [2, 17], sizes of subdomains are independent from the maximum mesh size allowed by the sequential remesher. In the following, we will provide further details on each of these five steps. A thorough description of all of the sketched algorithms can be found in [13].

3.1.1 Tagging of elements to be remeshed

The purpose of the first step is to determine the set of elements that need to be remeshed during the inner loop. Step 1a, which represents the initialization phase of the inner loop, yields the tags that have been computed at the beginning of the outer loop, without any further processing. In subsequent inner iterations, corresponding to step 1b, tags are removed from the elements that have been successfully remeshed. Elements which belong to the skin of the remeshed areas usually remain tagged, because the sequential remesher has been prevented from modifying them in order to reintegrate the remeshed areas.

3.1.2 Computation of independent pieces

This step aims at computing pieces of the mesh that can be remeshed independently from each other. Three constraints must be satisfied.

- Firstly, pieces must contain mostly elements that have been tagged. However, because remeshers often have to modify the neighboring elements of an element being refined, a piece may contain additional un-tagged elements that form a skin around the considered tagged elements.
- Secondly, the size of the pieces must be big enough to leave room for work to the sequential remesher. Yet, there must be enough pieces not to leave any PE idle. Hence, the size of the pieces may depend on the number of PEs. Also, it has to be bounded by the PE's available memory.
- Thirdly, the size of the frontier, that is, the number of elements that are adjacent to elements that do not belong to the piece, must be as small as possible, so that the aspect ratio of the piece to remesh is as high as possible, in order to constrain the remesher as little as possible in its operations. This third criterion may be evaluated using the isoperimetric quotient metric, which ranges from 0 to 1. A value of 1 indicates that the surface of the piece is minimized with respect to its volume, as it is the case for a sphere.

In the context of our algorithm, we consider the frontier of a piece, that is, the elements that are connected to elements that do not belong to the piece, and which should not be modified by the

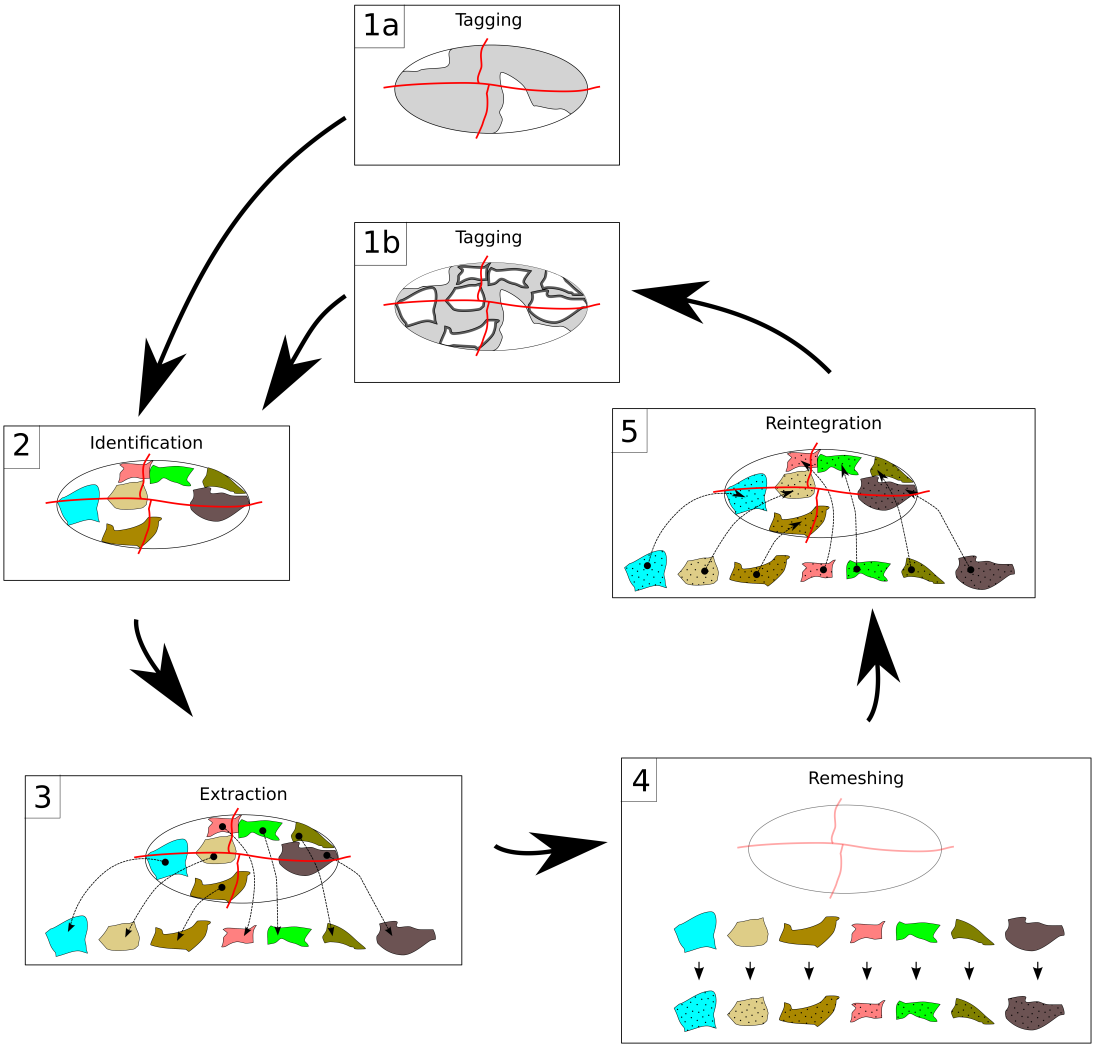


Figure 1: Sketch of the inner loop of our algorithm, showing its five steps. In the first step ($1a$, $1b$), elements that still need to be remeshed (the gray area) are tagged. In the second step, independent pieces are computed. In the third step, pieces are redistributed across PEs, each PE gathering all the data of a given piece, so as to balance the load of subsequent work. In the fourth step, sequential remeshing is concurrently performed on each PE. In the fifth, last step, remeshed pieces are reintegrated into the distributed mesh. This five-step loop iterates until no tagged element remains.

sequential remesher. In order to decrease the number of outer iterations, our aim is to minimize the number of these elements. Since this metric is not taken into account by parallel graph partitioners, we tried out several algorithms to obtain the pieces, and evaluated their output with respect to the frontier metric.

The most efficient method on average, for out test cases, consists in partitioning the dual

graph of the mesh. The desired number of parts is determined by estimating the number of nodes that will be created or removed by the remeshing process. Each of the tagged elements is assigned a weight that represents an estimation of the amount of work that the remesher will spend to process this element. The parallel partitioner PT-Scotch is used to partition this weighted dual graph into the desired number of pieces.

3.1.3 Extraction of the pieces to be remeshed

The pieces that have been computed at the previous step may span across multiple PEs. The goal of this step is therefore to gather all the fragments of a given piece to the same PE, so that PEs can work on centralized instances of the pieces that they have to remesh. All the entities adjacent to an element that belongs to some piece are copied to the destination PE for this piece. Consequently, some entities (*e.g.* nodes) may be duplicated across several PEs, if they are part of elements that belong to different pieces and which are to be processed by different PEs.

The assignment of pieces to PEs is also performed by using the Scotch partitioner. Its aim is to balance the estimated remeshing workload of the pieces across the PEs, while migrating as little data as possible across PEs, by favoring the assignment of pieces to PEs that already hold most, or at least some, of their data. To perform this assignment, a constrained partitioning is computed on a bipartite graph, as shown in figure 2.

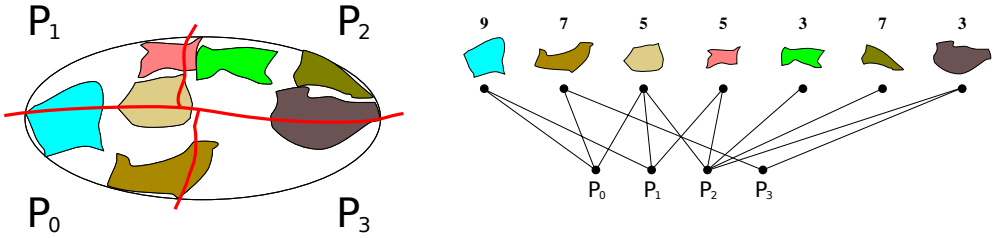


Figure 2: Bipartite graph used to extract pieces to PEs.

This graph comprises two sets of vertices: weighted vertices that represent the pieces, and fixed vertices that represent the PEs. A piece vertex is connected to a PE vertex if some of the data of the piece are originally stored on this PE. The weight of the edge depends on the amount of data on the PE. Hence, minimizing the cut of the partition of piece vertices amounts to minimizing data migration.

3.1.4 Remeshing of the pieces

Every piece, now on the form of a centralized submesh, is processed concurrently by an instance of the sequential, third-party remesher. In order to be able to reintegrate the remeshed pieces into the distributed mesh, frontier elements must never be modified by the remesher. Also, if a remesher instance fails when remeshing some piece, the original piece structure is preserved, with its tags remaining set, so as to be processed in a subsequent outer loop. Our framework is therefore resistant to remesher failure.

3.1.5 Reintegration of the remeshed pieces

Once the pieces have been remeshed, they have to be reintegrated into the unmodified part of the distributed mesh. This requires to merge the new topological information, as well as the new

values associated with the new mesh entities. The tags of elements which need to be remeshed but which belong to the skin of the pieces are kept, since these elements could not be remeshed. Once this step is complete, a new iteration of the inner loop can take place. After all the tagged elements have been remeshed, the new distributed mesh may be repartitioned and redistributed, so as to rebalance workload across the PEs.

Our parallel remeshing method has been implemented in the PaMPA parallel library. An interface has been defined, which allows one to use any sequential remesher that allows some of the elements of the mesh that is passed to it not to be modified.

3.1.6 Load balancing

The remeshing process, which may significantly modify the structure of the mesh, is likely to create load imbalance and/or communication overhead if remeshed mesh data is not properly redistributed. Redistribution can conveniently take place at two steps: during reintegration of the pieces and when the distributed mesh is built.

Load balancing during reintegration The load balancing strategy that we implemented at reintegration time is the following. When a piece comes entirely from a single PE, all the elements of the remeshed piece are sent back to this PE. When a piece is shared by more than one PE, the elements of the remeshed piece are distributed across the involved PEs, in the same proportion as they were distributed in the original piece. The rationale for this strategy is that if the mesh is to be uniformly remeshed, as it is the case for instance when creating a dense distributed mesh from some coarse mesh produced by a mesher, the increase in the number of elements is likely to be uniformly distributed.

In order to select the destination PE for each remeshed element, while minimizing communication at a local scale, the remeshed piece is partitioned into as many parts as there are involved PEs. Scotch is used to compute a constrained sequential mapping (and not only a partitioning), with fixed vertices, of the dual graph of the remeshed piece, as shown in Figure 3. Vertices that represent skin elements are fixed to their origin PE, while vertices that represent other elements, which belong to the interior of the piece, are not constrained. Then, a target architecture for mapping with Scotch is built, comprising as many target vertices as there are PEs sharing the piece. The capacity of each target vertices is set according to the ratio of elements located on each PEs before remeshing took place.

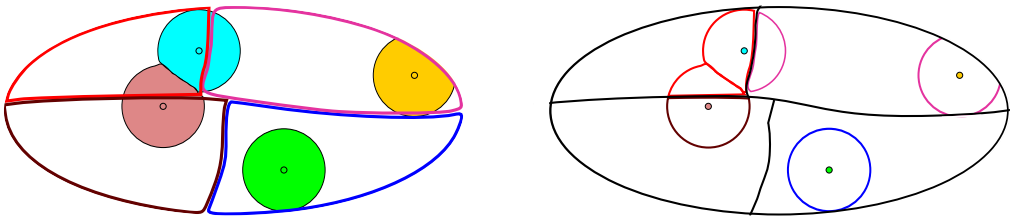


Figure 3: Constrained partitioning with fixed vertices to preserve the same ratio of elements accross PEs.

Load balancing after the new distributed mesh is built The previous local load balancing technique does not yield balanced meshed when remeshing is not uniform: a small group of

PEs may be overloaded by many newly created elements, or underloaded if the part of the mesh they own is coarsened. PE overloading not only creates computing inefficiency, but may also lead to remesher failure, when the memory of some PE gets full. Consequently, if, at the end of an iteration, the difference between the smallest and largest subdomains is higher than 25%, an additional, global repartitioning is performed, using PT-Scotch.Since, to date, parallel repartitioning is not available in PT-Scotch only parallel partitioning from scratch is performed.

3.2 Interfacing with the sequential remesher

Remeshing libraries are usually complex software. An advantage of our method is that it does not require profound changes in their code. Our only requirement is that these libraries should keep untouched a provided list of elements: those who correspond to the skin of the piece to remesh. To date, we have coupled our parallel software framework with Mmg3d, the 3D remesher of the mmg platform [9]. Mmg3d is an open-source software for simplicial remeshing based on local mesh modifications. A metric tensor field is provided at the vertices of a given triangulation. The remeshing process is controlled by two criteria: the boundary approximation and the metric field. An ideal surface model of the initial mesh is built, based on third-order Bézier patches. Then, the maximal distance between the piecewise linear representation of the boundary and this ideal surface is controlled along the surface remeshing process. The compliance of the remeshing with the size map is checked by checking edge lengths. Given a metric field prescription, the lengths of all mesh edges of the final optimized mesh should be as close as possible to that prescribed by the metric field, and mesh quality has to be as close as possible to the optimal unit value. Vertices are inserted in the volumic mesh using an anisotropic Delaunay kernel, and all the usual mesh operators (that is, collapse, edge swap, node relocation) are used.

3.3 Parallel interpolation

Currently, no interpolation is performed after parallel remeshing. Interpolation can take place at two different times: either sequentially at the end of each sequential remeshing task, before reintegrating the remeshed piece into the original mesh, or in parallel at the end of the whole remeshing process. On the one hand, with the first approach, several interpolations may be performed successively on the same area of the mesh, resulting in numerical inaccuracy. On the other hand, the second approach is very difficult to implement, because PEs may not own the same pieces of the mesh before and after remeshing and redistribution. Some global communication has to take place, in order to match the geometry of the remeshed mesh with that of the original mesh. For the sake of simplicity, we plan to follow the first approach. However, to mitigate the inaccuracy issue, we will favor as much as possible the production of large pieces, whose inner mesh elements will be more likely to reach optimal sizes in only one remeshing pass.

4 Experiments with an industrial test case

Mesh generation of the complex geometries encountered in industrial cases if often limited by the capabilities of meshing software and by the user's experience. Knowing where to set the highest resolution so as to get the most accurate results requires a lot of expertise. However, most meshing tools only provide global mesh refinement, where all the elements of the domain are scaled (up or down) to increase (or decrease) the discretization. Mesh adaptation is consequently a powerful tool allowing for targeted mesh refinement.

The industrial test case that we present in this report concerns a gas turbine (see Figure 4). This experiment took place as a follow-up to a study on the impact of mesh resolution impact on soot prediction in combustion [15]. In this study, large eddy simulation (LES) was used to predict the combustion characteristics of the system, including soot creation. Mesh discretization is a highly critical aspect of LES for capturing the physics of the problem, and mesh convergence is a critical diagnostic for the validity of the results. In this experiment, a standard quality mesh comprising 11 millions of tetrahedra was used as first guess. From this first mesh, three consecutive levels of refinement were generated, up to 1 billion of elements. The first level of refinement can usually be achieved via the standard meshing tools and global refinement. However, more detailed meshes are out of reach from standard tools, as memory and time become an issue. In this study, generating a first homogeneously refined mesh from the 33 million tetrahedra case up to 200 million elements required 7 hours to compute, and 7 more hours to dump to disk, with mixed quality results. As a more viable alternative, PaMPA was used, using selective refinement, to increase mesh resolution inside the elbow-shaped combustion chamber without modifying the refinement in the outer case zone where no combustion takes place, leading to a 220 million elements mesh (see Figure 5). Subsequently, PaMPA was used a second time to remesh the geometry further, so as to increase the resolution up to 1 billion elements in the same combustion region as before. This last refinement level is out of reach of current state-of-the-art meshing tools with acceptable time and resources constraints, that is, using less than 1 Tb of memory and taking less than 1 week for the meshing process. With PaMPA, the third level of remeshing took 10 minutes on 120 Intel E5-2680v4 Broadwell Xeon cores running at 2.40 GHz. To avoid I/O and memory bottlenecks, as each core can only address 4 GB of memory, partitioned input and output files were generated and coalesced using the `mdist` and `mcat` tools of the PaMPA distribution, respectively. The pre- and post-processing phases took 40 minutes each on a single Haswell core at 2.4 GHz.

Conclusion

The parallel remeshing framework implemented in PaMPA allowed us to process an industrial test case for a LES simulation that could not be tackled with prior state-of-the-art tools. The approach that we considered, which is based on distributed-memory, coarse-grained parallelism, needs to balance evenly remeshing workload across all PEs. This requires to estimate accurately enough the remeshing workload attached to each element, and to create as many independent pieces as there are PEs assigned to the user.

Although PaMPA has proved its usefulness and exhibits good scalability on thousands of PEs for large meshes [13], some of the algorithms implemented to date can be reworked in order to further improve parallel efficiency.

For instance, at the time being, element redistribution after remeshing is considered independently for each remeshed piece. This may result in overloading or underloading some PEs, which requires a subsequent, global redistribution step. A more efficient solution might be to consider element redistribution collectively on all pieces. Element redistribution would be performed in a localized way so as to share evenly the workload associated with the new elements, eliminating the need for subsequent global redistribution. Also, in order to reduce data movement and redistribution time, parallel repartitioning should be used instead of parallel partitioning from scratch. This requires an improvement of the PT-Scotch software.

Another slight improvement might be to consider vertex partitioning rather than edge partitioning, when creating the pieces to remesh. Indeed, with edge partitioning, two layers of untouched, skin elements are created between two pieces, which reduces piece sizes and thus

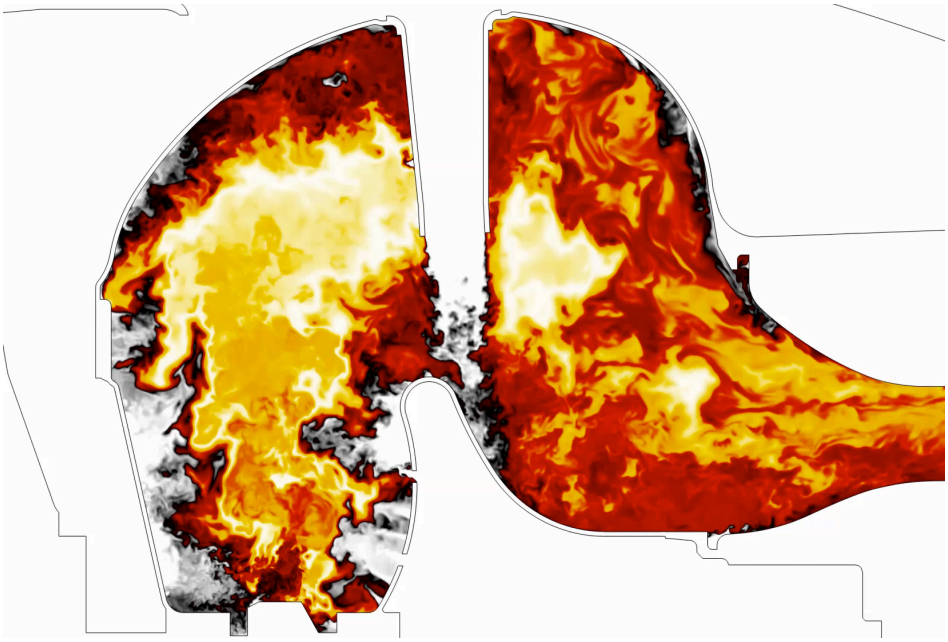


Figure 4: Cut view of the temperature field inside a gas turbine demonstrator.

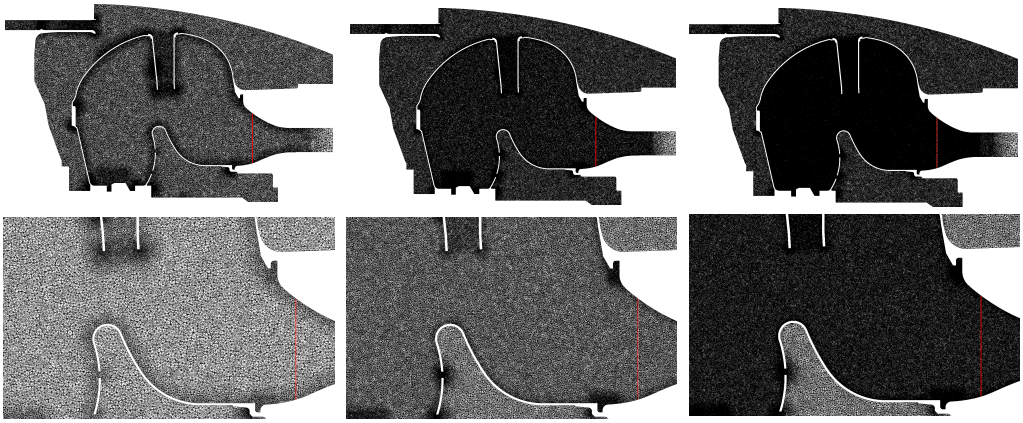


Figure 5: Top: increasing refinement of the mesh from 11 to 220 millions of elements. Bottom: detailed view

remeshing performance. With vertex partitioning, only one layer of skin elements would be left untouched, shared by the neighboring pieces.

Finally, PaMPA could be interfaced with more sequential remeshing tools. Some work has been carried out concerning Tetgen and GMSH, to enlarge the potential user base of PaMPA.

References

[1] A. Basermann, J. Clinckemaillie, T. Coupez, J. Fingberg, H. Digonnet, R. Ducloux, J.-M. Gratien, U. Hartmann, G. Lonsdale, B. Maerten, D. Roose, and C. Walshaw. Dynamic load-balancing of finite element applications with the drama library. *Applied Mathematical Modelling*, 25(2):83 – 98, 2000. Dynamic load balancing of mesh-based applications on parallel.

[2] P. Benard, G. Balarac, V. Moureau, C. Dobrzynski, G. Lartigue, and Y. D'Angelo. Mesh adaptation for large-eddy simulations in complex geometries. *International Journal for Numerical Methods in Fluids*, 2015.

[3] A. Casagrande, P. Leyland, L. Formaggia, and M. Sala. Parallel mesh adaptation. *Series on Advances in Mathematics for Applied Sciences*, 69:201, 2005.

[4] J. G. Castanos and J. E. Savage. The dynamic adaptation of parallel mesh-based computation. Citeseer, 1997.

[5] P. A. Cavallo, N. Sinha, and G. M. Feldman. Parallel unstructured mesh adaptation method for moving body applications. *AIAA journal*, 43(9):1937–1945, 2005.

[6] N. Chrisochoides and D. Nave. Simultaneous mesh generation and partitioning for delaunay meshes. *Mathematics and Computers in Simulation*, 54(4):321–339, 2000.

[7] N. Chrisochoides and D. Nave. Parallel delaunay mesh generation kernel. *International Journal for Numerical Methods in Engineering*, 58(2):161–176, 2003.

[8] T. Coupez, H. Digonnet, and R. Ducloux. Parallel meshing and remeshing. *Applied Mathematical Modelling*, 25(2):153–175, 2000.

[9] C. Dapogny, C. Dobrzynski, and P. Frey. Three-dimensional adaptive domain remeshing, implicit domain meshing, and applications to free and moving boundary problems. *Journal of Computational Physics*, 262(0):358–378, 2014.

[10] H. Digonnet, T. Coupez, P. Laure, and L. Silva. Massively parallel anisotropic mesh adaptation. *The International Journal of High Performance Computing Applications*, page 1094342017693906.

[11] C. Dobrzynski and J.-F. Remacle. Parallel mesh adaptation. *International Journal for Numerical Methods in Engineering*, 2007.

[12] P.-L. George and H. Borouchaki. *Delaunay triangulation and meshing: application to finite elements*. Hermes Paris, 1998.

[13] C. Lachat. *Conception et validation d'algorithmes de remaillage parallèles à mémoire distribuée basés sur un remailleur séquentiel*. PhD thesis, Université de Nice Sophia-Antipolis, 2013.

[14] I. L. Laemmer. Parallel mesh generation. In *Solving Irregularly Structured Problems in Parallel*, pages 1–12. Springer, 1997.

[15] J. Lamouroux, S. Richard, Q. Malé, G. Staffelbach, A. Dauptain, and A. Misdariis. In *ASME Turbo Expo 2017: Turbomachinery Technical Conference and Exposition*, page V04BT04A004; 10 pages, 2017. TE2017-64262.

[16] O. Lawlor, S. Chakravorty, T. Wilmarth, N. Choudhury, I. Dooley, G. Zheng, and L. Kalé. Parfum: a parallel framework for unstructured meshes for scalable dynamic physics applications. *Engineering with Computers*, 22:215–235, 2006. 10.1007/s00366-006-0039-5.

[17] A. Loseille, V. Menier, and F. Alauzet. Parallel generation of large-size adapted meshes. *Procedia Engineering*, 124:57–69, 2015.

[18] B. Maerten, D. Roose, A. Basermann, J. Fingberg, and G. Lonsdale. Drama: A library for parallel dynamic load balancing of finite element applications. In P. Amestoy, P. Berger, M. Daydé, D. Ruiz, I. Duff, V. Frayssé, and L. Giraud, editors, *Euro-Par'99 Parallel Processing*, volume 1685 of *Lecture Notes in Computer Science*, pages 313–316. Springer, 1999. 10.1007/3-540-48311-X_40.

[19] V. Menier. *Numerical methods and mesh adaptation for reliable rans simulations.* PhD thesis, Université Paris 6, 2015.

[20] S. N. Muthukrishnan, P. S. Shiakolas, R. V. Nambiar, and K. L. Lawrence. Simple algorithm for adaptive refinement of three-dimensional finite element tetrahedral meshes. *AIAA journal*, 33(5):928–932, 1995.

[21] L. Oliker and R. Biswas. Plum: Parallel load balancing for adaptive unstructured meshes. *Journal of Parallel and Distributed Computing*, 52(2):150–177, 1998.

[22] L. Oliker, R. Biswas, and H. N. Gabow. Parallel tetrahedral mesh adaptation with dynamic load balancing. *Parallel Computing*, 26(12):1583–1608, 2000.

[23] S.-J. Owen. A survey of unstructured mesh generation technology. In *IMR*, pages 239–267, 1998.

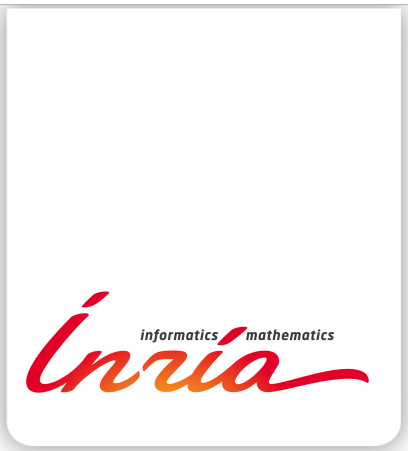
[24] M. Ramadan, L. Fourment, and H. Digonnet. A parallel two mesh method for speeding-up processes with localized deformations: application to cogging. *International Journal of Material Forming*, 2:581–584, 2009. 10.1007/s12289-009-0440-x.

[25] U. Tremel, K. A. Sørensen, S. Hitzel, H. Rieger, O. Hassan, and N. P. Weatherill. Parallel remeshing of unstructured volume grids for cfd applications. *International journal for numerical methods in fluids*, 53(8):1361–1379, 2007.

[26] N. A. Verhoeven, N. P. Weatherill, K. Morgan, et al. Dynamic load balancing in a 2d parallel delaunay mesh generator. In *Parallel Computational Fluid Dynamics*, pages 641–648, 1995.

[27] Pampa: Parallel mesh partioning and adaptation. `https://project.inria.fr/pampa`.

[28] Y. Yılmaz, C. Özturan, O. Tosun, A. H. Özer, and S. Soner. Parallel mesh generation, migration and partitioning for the elmer application.



RESEARCH CENTRE
BORDEAUX – SUD-OUEST

200 avenue de la Vieille Tour
33405 Talence Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399